

PROGRAMAÇÃO PARA DISPOSITIVOS MÓVEIS

Componentes Visuais
Gerenciadores de Layouts

Professor: Danilo Giacobbo



OBJETIVOS DA AULA

- Apresentar os conceitos básicos da programação de interfaces visuais para Android
- Apresentar os três componentes visuais básicos da plataforma Android
- Aprender a usar o componente de entrada de dados **EditText**
- Aprender a usar o componente de processamento de dados **Button**
- Aprender a usar o componente de saída de dados **TextView**
- Utilizar múltiplos gerenciadores de layout



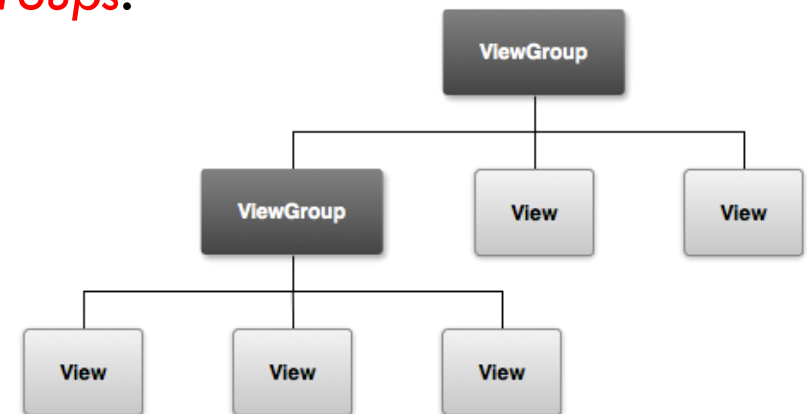
DESENVOLVIMENTO DE INTERFACES VISUAIS

- O Android oferece dois modos de criar interfaces para aplicativos móveis:
 - Definindo um arquivo XML que será carregado no início da aplicação, sendo que o desenho da tela é realizado durante a execução.
 - Codificando a interface dentro da classe Java semelhante aos aplicativos Java SE Swings ou AWTs.
- Na plataforma Android os componentes visuais são conhecidos como *Views*.
- Os gerenciadores de layouts são conhecidos como *ViewGroups*.

Lembre-se sempre que:

Arquivo XML: Interface Gráfica

Activity (classe Java): Lógica do Negócio



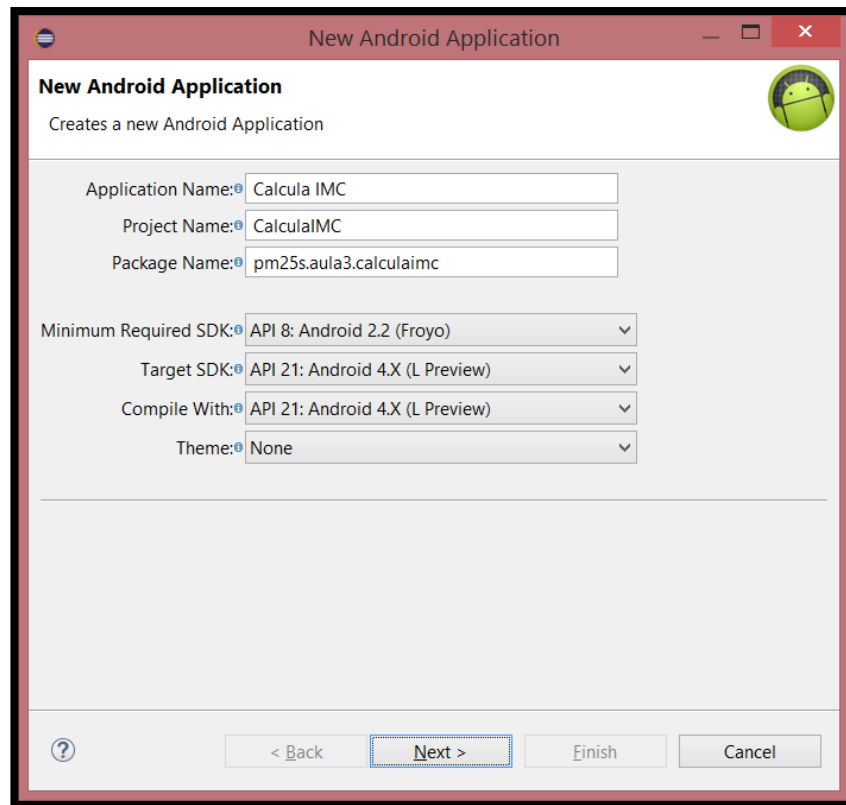
VIEW E VIEWGROUP - PRATICANDO...

- Este projeto visa exemplificar a utilização dos componentes visuais básicos do Android.
- O assunto deste projeto didático é o cálculo do IMC de uma pessoa.
- Ele possuirá dois componentes de entrada de dados **EditTexts**; um para o peso do indivíduo e outro para a sua altura.
- Também iremos inserir dois componentes do tipo **Button**; um para calcular o IMC e outro para limpar a entrada de dados.
- Para exibir o valor do IMC será utilizado um componente do tipo **TextView**.



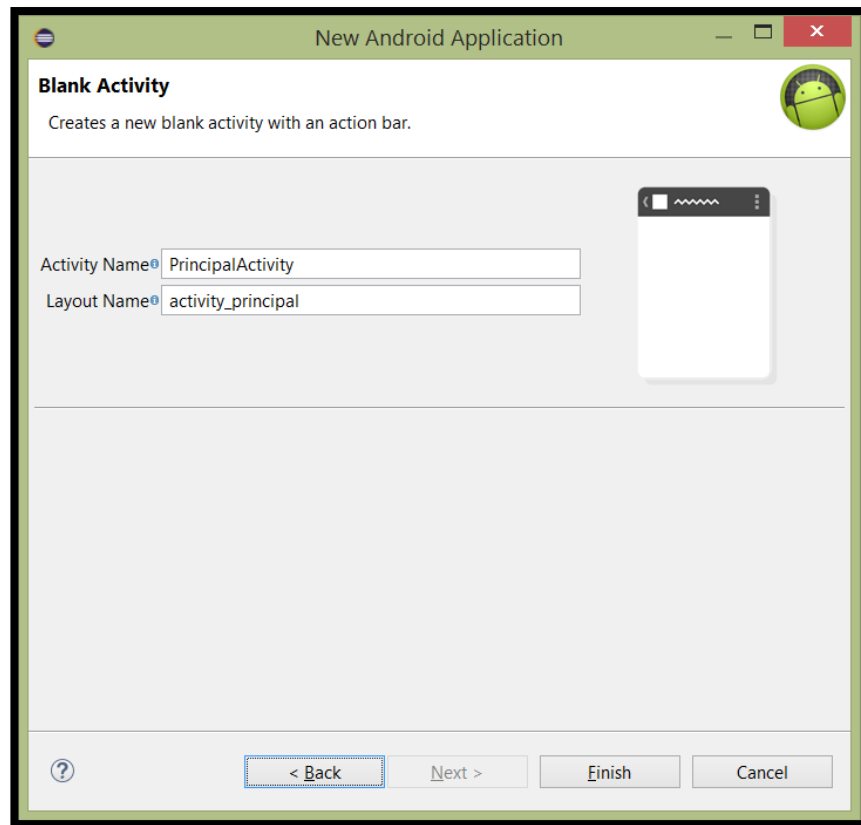
VIEW E VIEWGROUP - PRATICANDO...

- Crie um novo projeto Android chamado **CalculaIMC**.



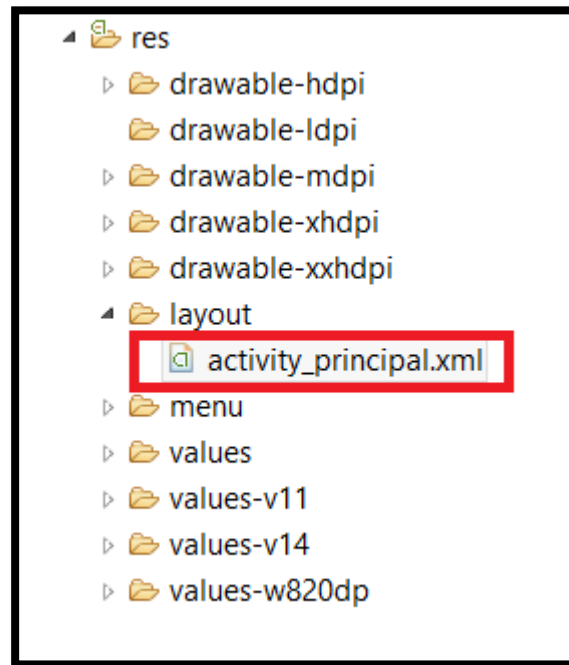
VIEW E VIEWGROUP - PRATICANDO...

- Tome cuidado ao nomear a sua **Activity**...



VIEW E VIEWGROUP - PRATICANDO...

- Vamos agora alterar a nossa interface gráfica. Clique duas vezes sobre o arquivo **activity_principal.xml**.



VIEW E VIEWGROUP - PRATICANDO...

- Dentro das tags `<LinearLayout>` e `</LinearLayout>` insira o código para mostrar o primeiro componente de entrada de dados (Peso da Pessoa) na tela do dispositivo:

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Peso: " />

<EditText
    android:id="@+id/etPeso"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:inputType="numberDecimal"
    android:text="" />
```



VIEW E VIEWGROUP - PRATICANDO...

- Dentro das tags `<LinearLayout>` e `</LinearLayout>` insira o código para mostrar o segundo componente de entrada de dados (Altura da Pessoa) na tela do dispositivo:

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Altura: " />

<EditText
    android:id="@+id/etAltura"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:inputType="numberDecimal"
    android:text="" />
```



VIEW E VIEWGROUP - PRATICANDO...

- Dentro das tags `<LinearLayout>` e `</LinearLayout>` insira o código para mostrar o resultado do cálculo do IMC da pessoa na tela do dispositivo:

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="IMC: " />

<TextView
    android:id="@+id/tvResult"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="0,0" />
```

VIEW E VIEWGROUP - PRATICANDO...

- Dentro das tags `<LinearLayout>` e `</LinearLayout>` insira o código para mostrar os dois botões de processamento na tela do dispositivo:

```
<Button
    android:id="@+id/btCalcular"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Calcular" />

<Button
    android:id="@+id/btLimpar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Limpar" />
```



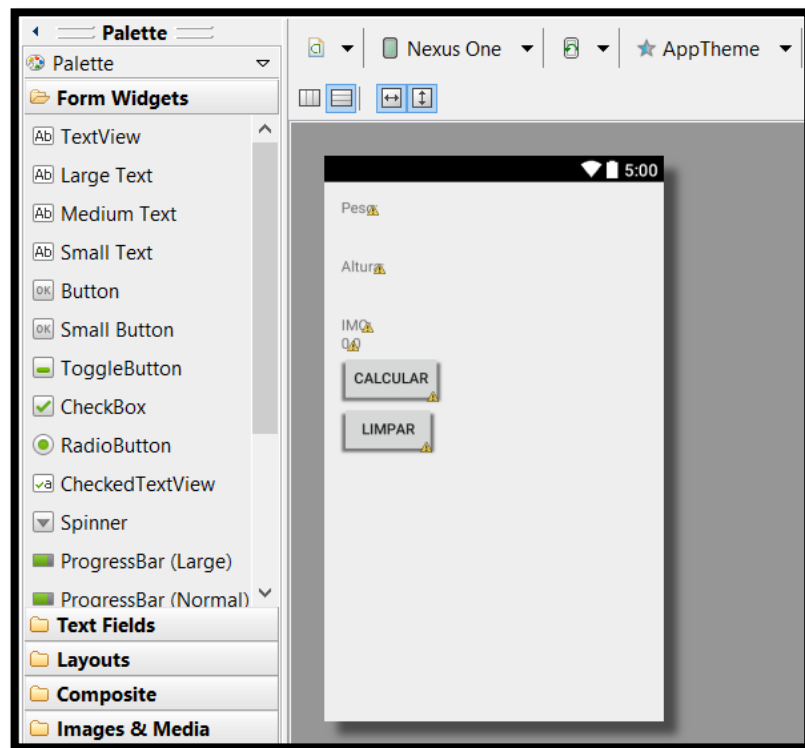
EXPLICANDO OS ATRIBUTOS XML DOS COMPONENTES

Atributo	Descrição
android:width	Esse atributo define a largura do componente visual na tela do dispositivo Android. Aconselha-se utilizar tamanhos relativos, como <i>fill_parent</i> (ocupa a largura total da tela do device) ou <i>wrap_content</i> (a largura se ajustará ao conteúdo no caso de um campo de texto, a largura do texto digitado no campo).
android:height	Esse atributo define a altura do campo. Aconselha-se também o uso do <i>fill_parent</i> (a altura total da tela do dispositivo) ou <i>wrap_content</i> (a altura será dinâmica, de acordo com o conteúdo que existe no campo).
android:text	O texto inicial do componente. Esta propriedade existe em todos os componentes que apresentam texto para o usuário, como os EditText e os TextView .
android:id	Este atribuirá um nome ao componente para que o mesmo possa ser referenciado durante a execução, via código-fonte, ou ser referenciado por meio de outro componente visual, via XML.

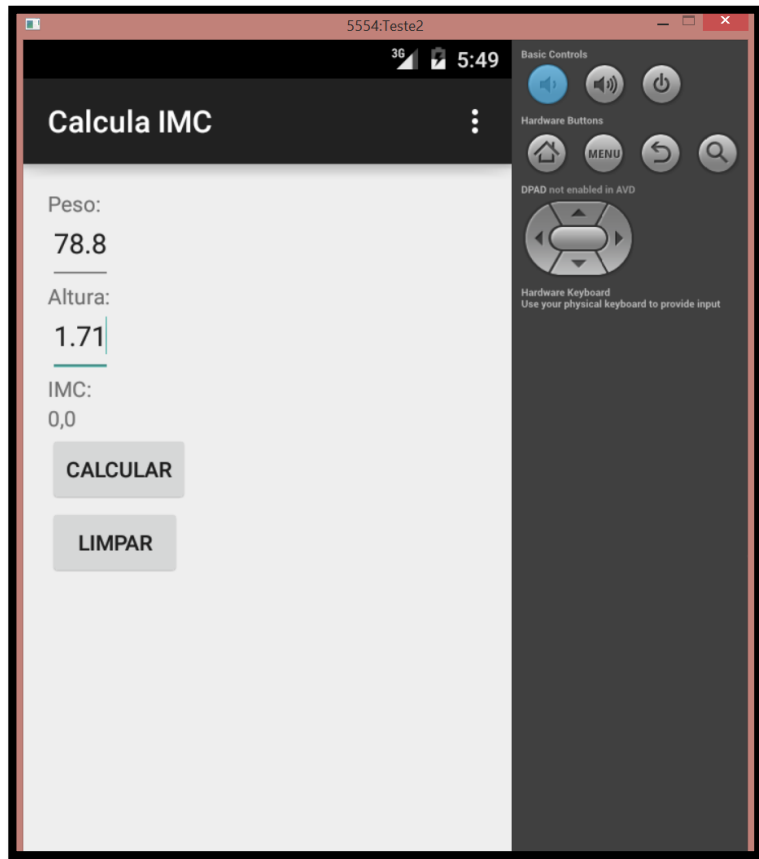


MODO GRAPHICAL LAYOUT DA INTERFACE

- Usando a opção **Graphical Layout** (a partir do arquivo XML) você poderá ver como os componentes foram dispostos na interface gráfica do dispositivo.



FILL_PARENT E WRAP_CONTENT



Botões com a largura como *wrap_content*



Botões com a largura como *fill_parent*

O COMPONENTE **EDITTEXT**

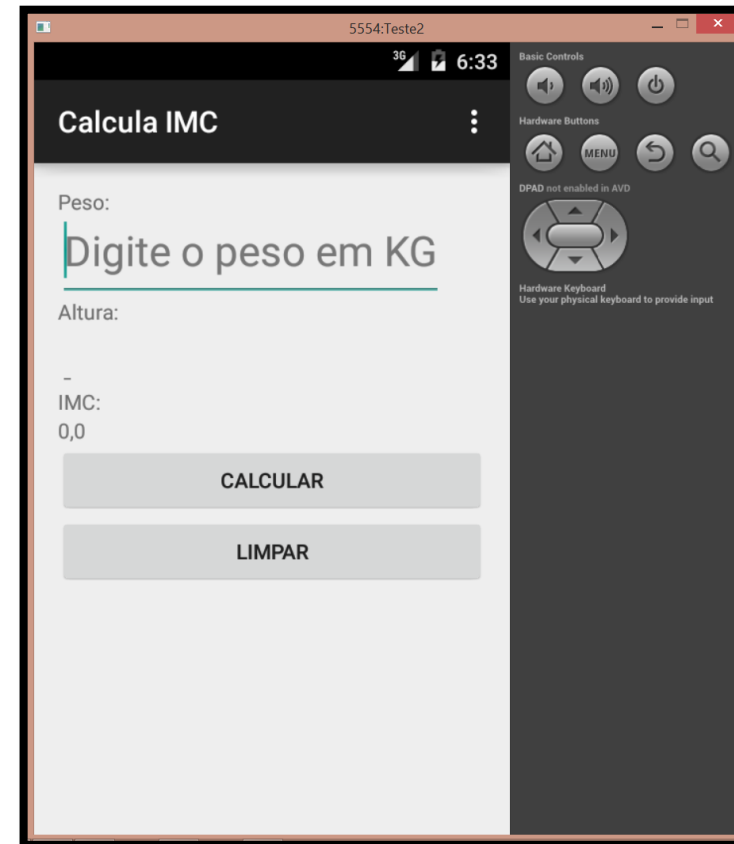
- É uma caixa de texto para a digitação de informação na tela do dispositivo. Entre as principais propriedades do **EditText** estão:

Atributo	Descrição
android:input_type	Define qual o tipo de informação que o EditText está preparado para receber. Pode ser: <ul style="list-style-type: none">- <u>number</u>: para número inteiros positivos- <u>numberDecimal</u>: para números com casas decimais Há vários outros tipos de entrada disponíveis como senhas, palavras iniciadas com letra maiúscula, todas em maiúsculas, etc.
android:max_length	Permite definir o limite de caracteres aceitos em um componente EditText , limitando, assim, a entrada de dados.
android:hint	Esta propriedade aceita um conjunto de caracteres que será apresentado dentro do componente EditText, quando vazio. É utilizado como uma dica de preenchimento, por exemplo.
android:textSize	Todo componente visual que apresenta texto possui esta propriedade, que define o tamanho do texto apresentado na tela.



O COMPONENTE **EDITTEXT**

```
<EditText
    android:id="@+id/etPeso"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:inputType="numberDecimal"
    android:hint="Digite o peso em KG"
    android:maxLength="8"
    android:textSize="28sp"
    android:text="" />
```



O COMPONENTE **TEXTVIEW**

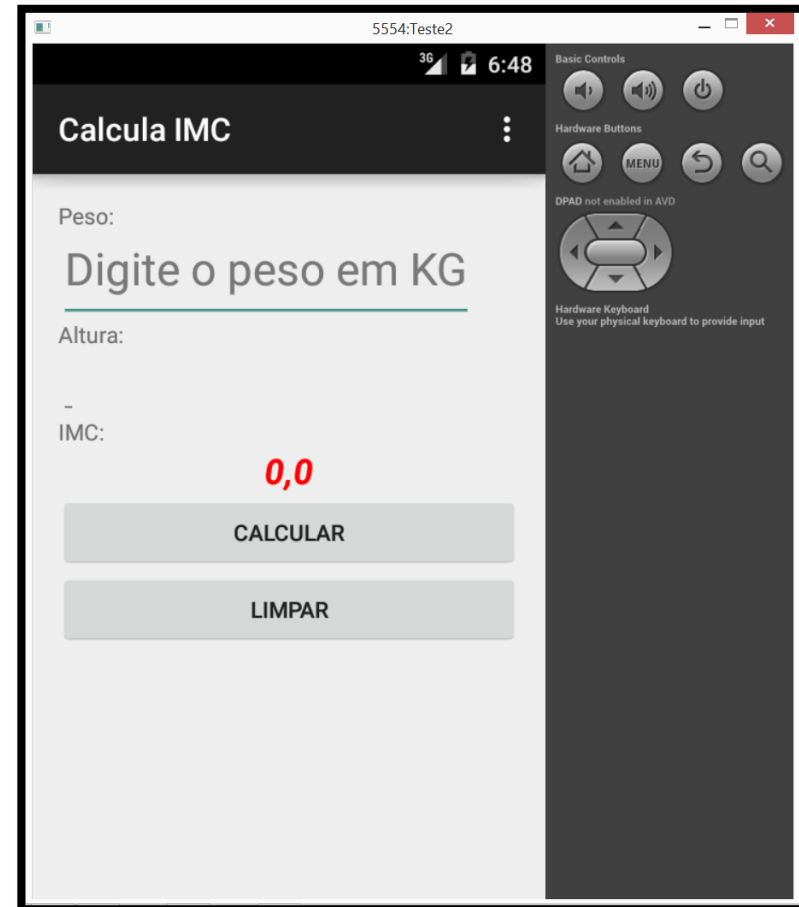
- É um componente que apresenta informações textuais estáticas para o usuário. Entre as principais propriedades do **TextView** estão:

Atributo	Descrição
android:textAppearance	Configura alguns estilos pré-definidos para um componente de texto.
android:textColor	Permite definir a cor do texto de um componente. Deve possuir um código hexadecimal de 6 dígitos iniciado com o sustenido (#).
android:typeFace	Permite escolher uma fonte para o texto. Como o Android é baseado no Linux, não existem muitas opções de fontes, destacando-se “monospace”, “serif” e “sans”.
android:textStyle	Esta propriedade permite definir o estilo do texto, podendo ser negrito ou itálico. Desejando mais de uma propriedade, as mesmas podem ser concatenadas usando o caractere . Ex.: <code>android:textStyle="bold italic"</code> .
android:layout_gravity	Esta propriedade define o alinhamento do texto, caso o componente visual ocupe a largura inteira da tela (ex.: <code>fill_parent</code>). Para alinhar centralizado o texto, por exemplo, pode-se utilizar “ <code>center_horizontal</code> ”.



O COMPONENTE **TEXTVIEW**

```
<TextView
  android:id="@+id/tvResult"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:textAppearance="?android:attr/textAppearanceLarge"
  android:textColor="#FF0000"
  android:typeface="monospace"
  android:layout_gravity="center_horizontal"
  android:textStyle="bold|italic"
  android:text="0,0" />
```



O COMPONENTE **BUTTON**

- É um componente interativo da plataforma, permitindo a ação de clique no mesmo e a execução de um código associado a este clique. Entre as principais propriedades estão:

Atributo	Descrição
android:enabled	Propriedade booleana, em que se pode habilitar ou não o componente Button para o clique.
android:focusable	Em dispositivos que possuem um <i>soft button</i> para alterar o foco entre diferentes componentes, se <i>focusable</i> estiver definido com <i>false</i> , este não receberá o foco.
android:onClick	Pode-se associar o clique do botão a algum método presente dentro da classe <i>Activity</i> . Essa associação se dá a partir da propriedade <i>onClick</i> .



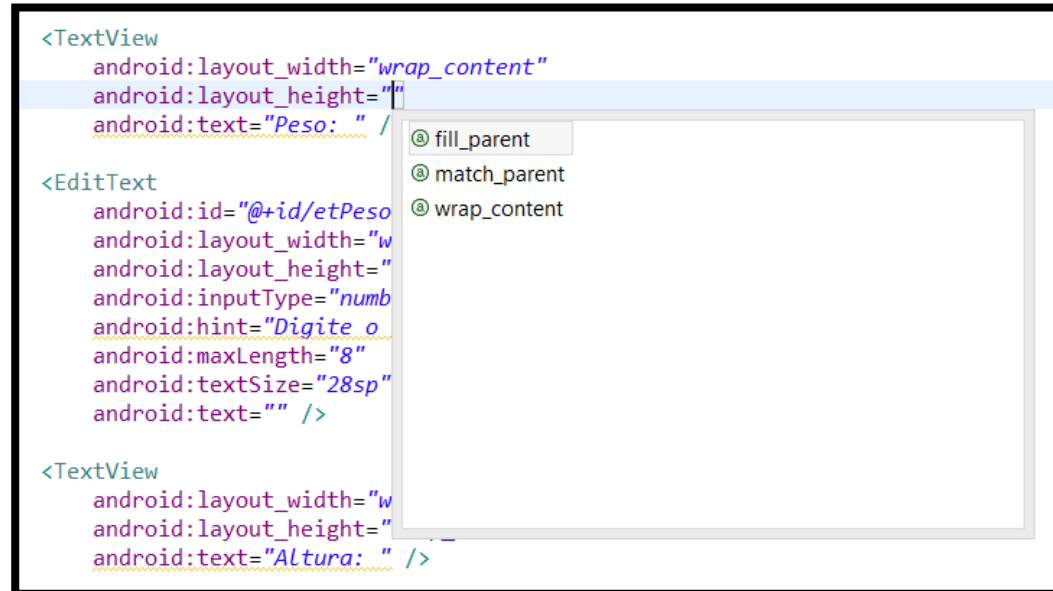
O COMPONENTE **BUTTON**

```
<Button  
    android:id="@+id/btCalcular"  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:text="Calcular"  
    android:enabled="false"  
    android:focusable="false"  
    android:onClick="btCalcularOnClick" />
```



DICA: ASSISTENTE DE CÓDIGO

- No editor de código da maioria dos IDEs, existem assistentes de código, podendo ser associados pela combinação de teclas [CTRL] + [ESPAÇO] no momento da digitação. Assim, para saber quais são os possíveis **input_types** do componente **EditText**, basta pressionar esta sequência de passos e escolher o desejado, conforme apresentado na figura abaixo:



```
<TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Peso: " />

<EditText
  android:id="@+id/etPeso"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:inputType="number"
  android:hint="Digite o peso"
  android:maxLength="8"
  android:textSize="28sp"
  android:text="" />

<TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Altura: " />
```

fill_parent
match_parent
wrap_content



CARACTERÍSTICAS DO **LINEAR LAYOUT**

- É o mais utilizado no desenvolvimento de aplicações móveis Android.
- Seu arranjo de componentes na tela é simples.
- Não depende do tamanho do *display* do dispositivo móvel.
- É um gerenciador de layout **relativo**.
- Cada componente é alocado na tela levando em consideração a posição do componente anterior.
- O próprio Google indica que os desenvolvedores preferem o uso de gerenciadores de layout relativos.



CARACTERÍSTICAS DO **LINEAR LAYOUT**

Calcula IMC

Peso:
Digite o peso em KG

Altura:
-

IMC:
0,0

CALCULAR

LIMPAR

Gerenciador LinearLayout vertical

Calcula IMC

Peso: | Altura: | IMC: **0,0** | CALCULAR

Gerenciador LinearLayout horizontal



LINEAR LAYOUT COM SCROLLVIEW

- Este gerenciador é utilizado nas interfaces visuais formadas por vários componentes em situações nas quais eles não cabem na tela do dispositivo móvel, assim, o **ScrollView** apresenta barras de rolagem para poder navegar pelos componentes.
- Basicamente, esse gerenciador de layout deve conter o gerenciador que necessitará de rolagem.
- Nós vamos agora alterar o nosso projeto de cálculo de IMC para contemplar esse gerenciador de layout novo.
- Existem outros gerenciadores que podem ser usados pelos aplicativos Android, tais como, **RelativeLayout**, **AbsoluteLayout**, **TableLayout**, entre outros que veremos ao longo das aulas.



LINEAR LAYOUT COM SCROLLVIEW

- No começo do arquivo XML com o código de layout da aplicação será necessário incluir as seguintes linhas:

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:paddingBottom="@dimen/activity_vertical_margin"
  android:paddingLeft="@dimen/activity_horizontal_margin"
  android:paddingRight="@dimen/activity_horizontal_margin"
  android:paddingTop="@dimen/activity_vertical_margin"
  tools:context="pm25s.aula3.calculaimc.PrincipalActivity" >

  <LinearLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
```



LINEAR LAYOUT COM SCROLLVIEW

- Vamos incluir mais alguns botões para que as barras de rolagem possam ser testadas:

```
<Button
    android:id="@+id/btExtra1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Extra 1" />
```

```
<Button
    android:id="@+id/btExtra2"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Extra 2" />
```

```
<Button
    android:id="@+id/btExtra3"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Extra 3" />
```

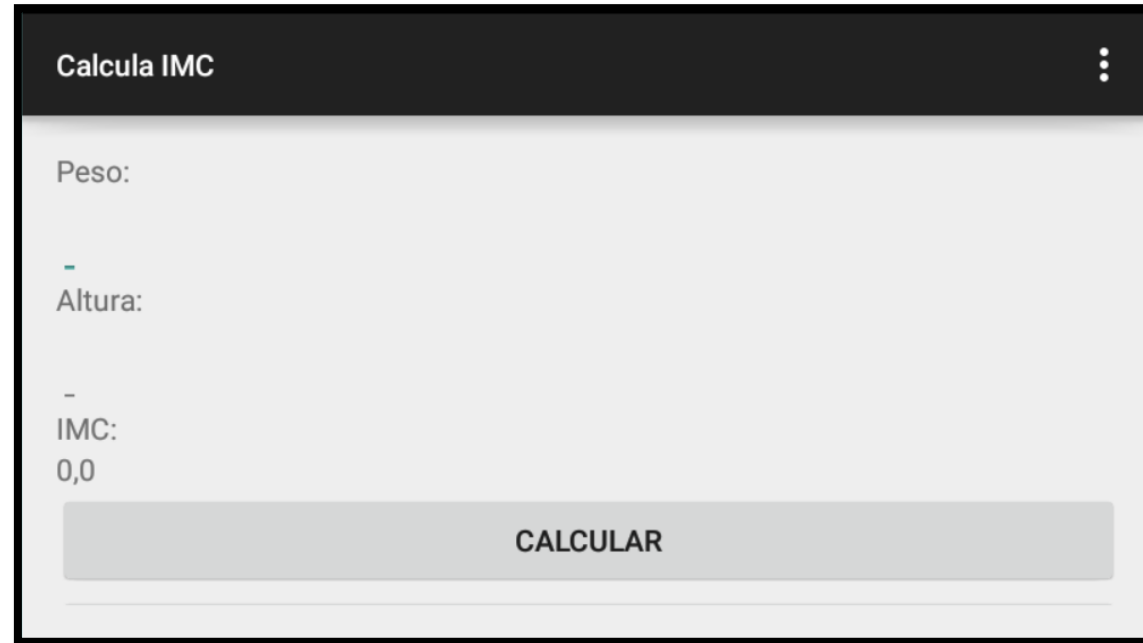
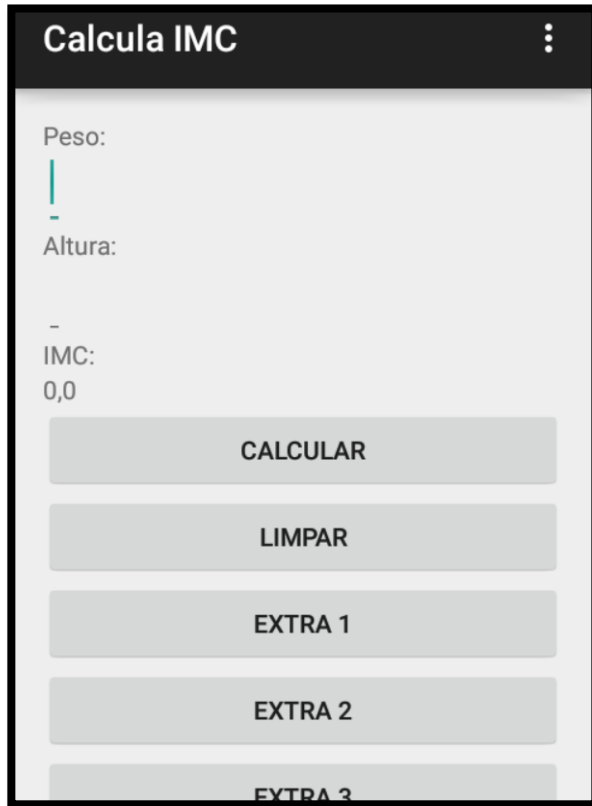
```
<Button
    android:id="@+id/btExtra4"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Extra 4" />
```

```
<Button
    android:id="@+id/btExtra5"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Extra 5" />
```

```
</LinearLayout>
</ScrollView>
```



LINEAR LAYOUT COM SCROLLVIEW



Exemplos de telas com vários componentes visuais e barra de rolagem



UTILIZANDO DOIS **LINEAR LAYOUTS**

- Para exemplificar o uso de dois gerenciadores de layout **LinearLayout** em uma mesma interface iremos alterar o nosso projeto do IMC para que os botões sejam apresentados um ao lado do outro enquanto que os demais campos permanecem um abaixo do outro.

```
<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal" >

    <Button
        android:id="@+id/btCalcular"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Calcular" />

    <Button
        android:id="@+id/btLimpar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Limpar" />

</LinearLayout>
```

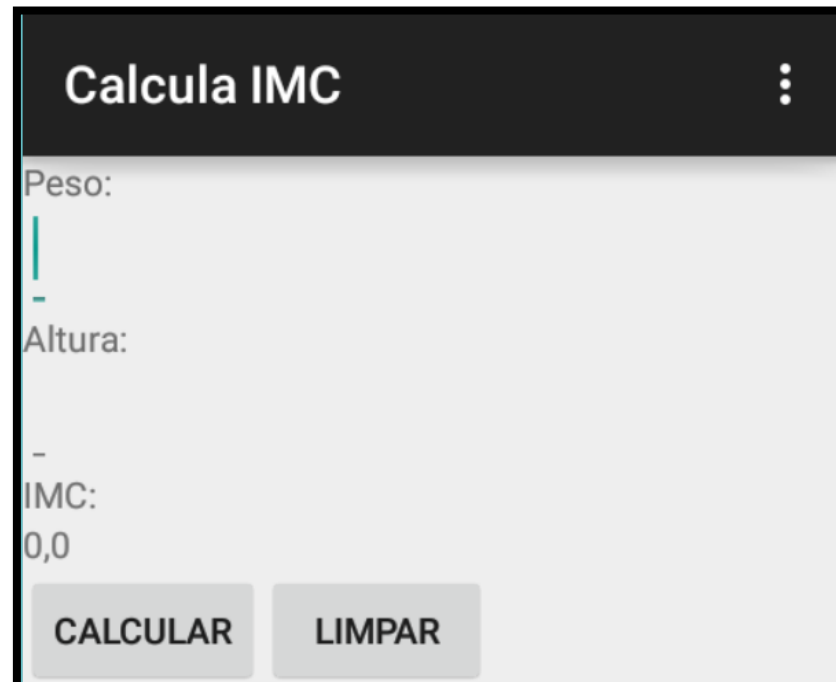


TABLE LAYOUT

- Este gerenciador é muito interessante para organizar os componentes no formato linha/coluna, sendo um dos mais utilizados para apresentar formulários de cadastros em aplicativos móveis.
- As linhas da tabela são representadas pela classe `TableRow`. As colunas são criadas dinamicamente à medida que se adicionam componentes visuais às linhas.
- No slide seguinte é apresentado um exemplo de XML que mostra o uso do `TableLayout`.
- O layout representado pelo código XML do slide seguinte faz uso de `TableLayout`, possuindo três linhas (`TableRow`). Cada linha é composta por dois componentes visuais.
- Observe que a largura dos componentes foi definida em *pixel* apenas na primeira linha. Neste layout a largura da coluna é definida de acordo com a largura do maior componente adicionado à tela.



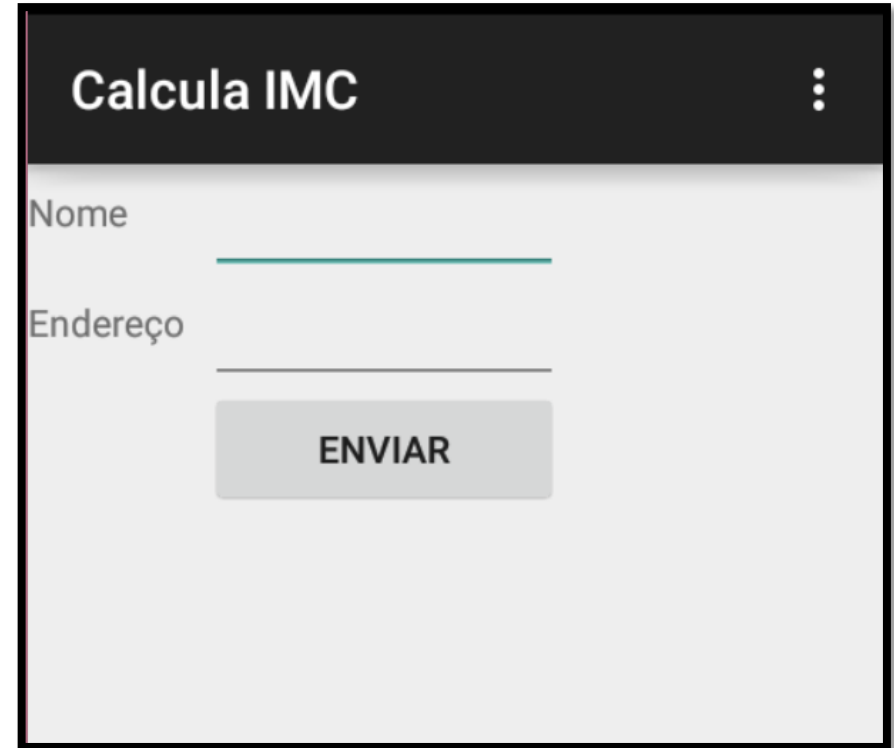
TABLE LAYOUT - EXEMPLO

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <TableRow>
        <TextView
            android:text="Nome"
            android:layout_width="100px" />
        <EditText
            android:id="@+id/EditText01"
            android:layout_width="200px" />
    </TableRow>

    <TableRow>
        <TextView android:text="Endereço" />
        <EditText android:id="@+id/EditText02" />
    </TableRow>

    <TableRow>
        <TextView />
        <Button android:id="@+id/Button01"
            android:text="Enviar" />
    </TableRow>
</TableLayout>
```



RELATIVE LAYOUT

- Este tipo de gerenciador permite adicionar componentes à tela, sendo que a posição destes leva em consideração a posição dos outros componentes adicionados ou do gerenciador de layout no qual ele encontra.
- Utilizando esse gerenciador, cada componente pode ficar em posições específicas na tela.
- Um exemplo de código XML deste tipo de layout e sua respectiva tela podem ser vistos no próximo slide.

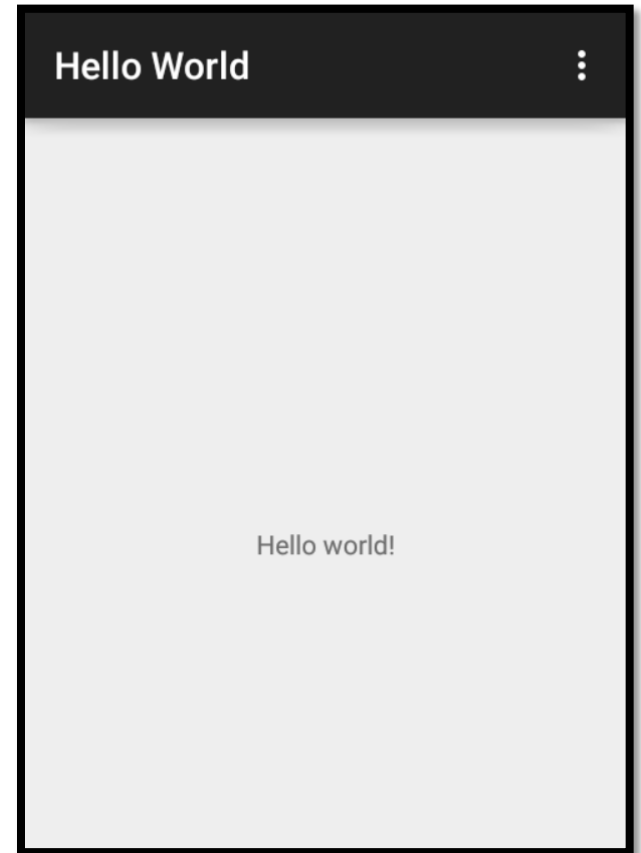


RELATIVE LAYOUT - EXEMPLO

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="pm25s.aula2.helloworld.PrincipalActivity" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="@string/hello_world" />

</RelativeLayout>
```



ABSOLUTE LAYOUT

- Este tipo de gerenciador define a posição dos componentes com base nas coordenadas **x** e **y** da tela.
- A grande vantagem desse layout é que se pode definir a posição exata em que os componentes ficarão.
- Um exemplo de código XML deste tipo de layout e sua respectiva tela podem ser vistos no próximo slide.
- Como pode ser observado os componentes visuais que se encontram em um **AbsoluteLayout** possuem dois novos atributos: **android:layout_x** e **android:layout_y**.
- Ele possui uma grande desvantagem: é pouco flexível e exige um grande trabalho para a manutenção da interface.



ABSOLUTE LAYOUT - EXEMPLO

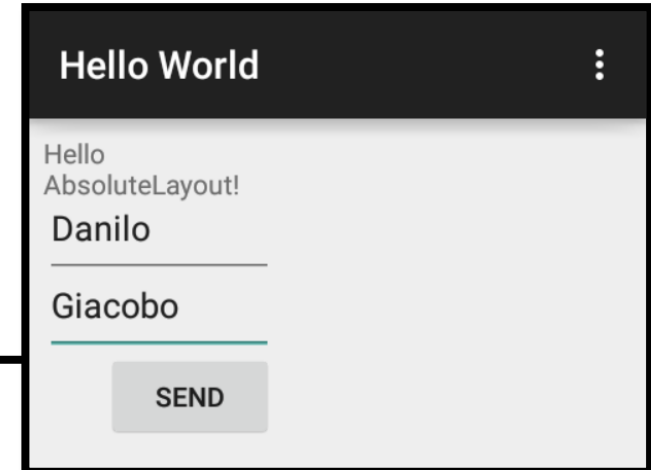
```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <TextView
        android:layout_width="180px"
        android:layout_height="wrap_content"
        android:text="Hello AbsoluteLayout!"
        android:layout_x="12px"
        android:layout_y="12px" />

    <EditText
        android:id="@+id/EditText01"
        android:layout_width="180px"
        android:layout_height="wrap_content"
        android:layout_x="12px"
        android:layout_y="60px" />
```

```
<EditText
    android:id="@+id/EditText02"
    android:layout_width="180px"
    android:layout_height="wrap_content"
    android:layout_x="12px"
    android:layout_y="120px" />

<Button
    android:id="@+id/Button01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Send"
    android:layout_x="60px"
    android:layout_y="180px" />
</AbsoluteLayout>
```



OUTROS GERENCIADORES DE LAYOUT

- A plataforma Android ainda possui outros gerenciadores de layout. Entretanto, com os gerenciadores apresentados anteriormente, já é possível desenvolver interfaces relativamente complexas.
- Dos gerenciadores de layout ainda não apresentados, destacam-se:
 - **FrameLayout**: este gerenciador de layout é utilizado para reservar um espaço na tela que será utilizado por um ou mais componentes. Se mais de um componente for colocado dentro de um único FrameLayout, haverá sobreposição, com o último componente inserido aparecendo sobre o primeiro.
 - **GridLayout**: adicionado recentemente, funciona de forma semelhante ao TableLayout. A principal diferença é que, neste caso, podem-se definir células vazias ou definir que uma célula ocupará mais de uma linha e/ou coluna. Obs.: As bibliotecas de compatibilidade permitem seu uso também nas versões mais antigas do sistema operacional.



CRIANDO INTERFACES A PARTIR DO CÓDIGO JAVA

- Embora seja pouco utilizado, é possível criar interfaces a partir do código-fonte da aplicação, desenvolvido em Java (classe *Activity*). Nela, é possível instanciar componentes visuais (**View**) e gerenciadores de Layout (**ViewGroup**) associando-os e, por fim, adicionando-os à tela a partir do comando **setContentView**.
- Assim, no lugar de criar o arquivo XML, o programador pode apenas usar o arquivo **PrincipalActivity.java**, conforme apresentado no próximo slide.
- Nesta situação, o arquivo **activity_principal.xml** pode ser excluído do projeto.



CRIANDO INTERFACES A PARTIR DO CÓDIGO JAVA

```
1 package pm25s.aula3.javaactivity;
2
3 import android.support.v7.app.ActionBarActivity;
4 import android.os.Bundle;
5 import android.view.Menu;
6 import android.view.MenuItem;
7 import android.text.InputType;
8 import android.widget.Button;
9 import android.widget.EditText;
10 import android.widget.LinearLayout;
11 import android.widget.TextView;
12
13 public class MainActivity extends ActionBarActivity {
14
15     @Override
16     protected void onCreate(Bundle savedInstanceState) {
17         super.onCreate(savedInstanceState);
18
19         LinearLayout linearVertical = new LinearLayout(this);
20         linearVertical.setOrientation(LinearLayout.VERTICAL);
21
22         TextView tvPeso = new TextView(this);
23         tvPeso.setText("Peso: ");
24
25         EditText etPeso = new EditText(this);
26         etPeso.setInputType(InputType.TYPE_NUMBER_FLAG_DECIMAL);
27
28         TextView tvAltura = new TextView(this);
29         tvAltura.setText("Altura: ");
30
31         EditText etAltura = new EditText(this);
32         etAltura.setInputType(InputType.TYPE_NUMBER_FLAG_DECIMAL);
```

```
34         TextView tvRotuloResultado = new TextView(this);
35         tvRotuloResultado.setText("Resultado: ");
36
37         TextView tvResultado = new TextView(this);
38         tvResultado.setText("0,0");
39
40         LinearLayout linearHorizontal = new LinearLayout(this);
41         linearHorizontal.setOrientation(LinearLayout.HORIZONTAL);
42
43         Button btCalcular = new Button(this);
44         btCalcular.setText("Calcular");
45
46         Button btLimpar = new Button(this);
47         btLimpar.setText("Limpar");
48
49         linearVertical.addView(tvPeso);
50         linearVertical.addView(etPeso);
51         linearVertical.addView(tvAltura);
52         linearVertical.addView(etAltura);
53         linearVertical.addView(tvRotuloResultado);
54         linearVertical.addView(tvResultado);
55
56         linearHorizontal.addView(btCalcular);
57         linearHorizontal.addView(btLimpar);
58
59         linearVertical.addView(linearHorizontal);
60         setContentView(linearVertical);
61
62     }
```

